OXFORD

Journal of Geophysics and Engineering

Journal of Geophysics and Engineering (2022) **19**, 389–402 https://doi.org/10.1093/jge/gxac028

Accelerating the 2+2+1 method for estimating local traveltime operators in nonlinear beamforming using GPU graphics cards

Yimin Sun [1,*], **Ilya Silvestrov**[2] **and Andrey Bakulin**[2]

[1] Aramco Research Center - Delft, Aramco Europe, Delft, 2628 ZD, The Netherlands
[2] EXPEC Advanced Research Center, Saudi Aramco, Dhahran, 31311, Saudi Arabia

*Corresponding author: Yimin Sun. E-mail: sun.delft@gmail.com

Received 13 December 2021, revised 21 March 2022
Accepted for publication 13 April 2022

Abstract

Local traveltime operators are an effective way to describe local kinematic wavefronts. They are useful for many applications. One of them is nonlinear beamforming for enhancing the signal-to-noise ratio of challenging seismic data. The so-called 2+2+1 method is a pragmatic approach to estimate unknown local traveltime operators from input data. However, its efficiency still has much room for improvement when the solution space is big. We accelerate the 2+2+1 method using graphics processing unit (GPU) computing with the Compute Unified Device Architecture (CUDA) programming language. We detail the CPU- and GPU-based 2+2+1 search algorithms and demonstrate the efficiency improvement using synthetic and field data examples. Compared to a standard multi-core CPU implementation, our new GPU implementation achieves almost the same quality results at only ∼10% run-time cost.

Keywords: GPU, signal restoration, geophysical signal processing, approximation algorithm, parallelization

1. Introduction

Seismic prospecting is a powerful tool in oil and gas exploration for retrieving subsurface geological information, and the related seismic data acquisition schemes are evolving all the time. Bakulin *et al.* (2020) introduced the current trend for modern three-dimensional (3D) land seismic data acquisitions. As better wavefield sampling can potentially benefit subsequent data processing, high-channel-count surveys or even single-sensor surveys are more and more preferred over the traditional scheme of acquiring sparse data with large field arrays. However, a significant disadvantage of using denser grids of smaller receiver arrays is the very low signal-to-noise ratio (SNR) of raw data. Figure 1 in Bakulin *et al.* (2018) is a representative comparison of the raw data quality between a traditional data acquisition scheme and a modern scheme. As of today, it is still a general technical challenge for the seismic industry to deal with modern 3D land seismic data properly.

Nonlinear beamforming (NLBF) is a recently proposed technology to improve the quality of modern 3D seismic data. The complete theory behind NLBF has been discussed in Bakulin *et al.* (2018, 2020). In a nutshell, it is a two-step process: in the first step, local traveltime operators of NLBF parameter traces are estimated by maximizing a semblance-based cost function; next, every trace in the original dataset is stacked with its nearby traces via the guidance of the estimated local traveltime operators for SNR improvement. To some extent, NLBF is similar to some other traveltime-operator-based technologies in the field of seismics, such as the Common Focus Point technology (Hindriks & Duijndam 1999; Sun *et al.* 2014), the Common Reflection Surface (CRS) technology (Zhang *et al.* 2001;

Jäger *et al.* 2001) and the multifocusing technology (Berkovitch *et al.* 2011). However, NLBF stands out from the other algorithms in its flexibility: NLBF can work in different data domains, such as shot-gather domain, receiver-gather domain, cross-spread domain, etc.; NLBF can also be applied to data after normal moveout corrections (NMO).

Estimating local traveltime operators from an input seismic dataset is very challenging and time-consuming. As discussed in Sun *et al.* (2022), the target function involved in estimating these operators is highly nonlinear, so an efficient solver is much needed. Commonly seen and used global optimization solvers, such as simulated annealing (Metropolis *et al.* 1953; Ingber 1996), differential evolution (Storn and Price 1997) and genetic algorithms (Sun and Verschuur 2014; Sun *et al.* 2016; Sun *et al.* 2017; Acuna and Sun 2020), could be potential candidates for this nonlinear problem. Recently we have demonstrated the success of applying the efficiency-improved genetic algorithm to estimate local traveltime operators for NLBF (Sun *et al.* 2022). As a pragmatic solution, it is also a common practice to use local-search solvers for this nonlinear problem. The 2+2+1 method is such a solver to estimate local traveltime operators (Bakulin *et al.* 2018). It has been successfully used in real data processing (Bakulin *et al.* 2020; Bakulin *et al.* 2021). Note that for the CRS technology, a pragmatic local-search solver, similar to the 2+2+1 method, is also in existence and often used in practice (Jäger *et al.* 2001).

Since the launch of the Compute Unified Device Architecture (CUDA) programming language in 2007, graphics processing unit (GPU) computing has become prevalent in high-performance computing. In addition, more and more supercomputers today are powered by GPUs ("Top 500", 2021) as they are more energy-efficient than traditional supercomputers powered only by CPUs. GPU computing has found many successful applications so far, such as fluid dynamics simulation (Elsen *et al.* 2008), finite-element simulation (Klöckner *et al.* 2009), reverse-time migration (Shi & Wang 2016; Li *et al.* 2018), geostatistical inversion (Liu & Grana 2019), and so on. Nowadays, it is common to have an application-software package implemented for both CPU and GPU platforms simultaneously in the seismic industry.

This paper introduces our work in accelerating the 2+2+1 method for estimating local traveltime operators in NLBF using GPU computing via the CUDA programming language. The rest of this paper is organized as follows: we first briefly introduce the local traveltime operators; next, we detail the 2+2+1 method and its algorithmic implementations for both CPU and GPU platforms; finally, by using both a synthetic dataset and a challenging field dataset, we demonstrate the efficiency gain of our GPU implementation over the CPU version on estimating local traveltime operators for NLBF.

2. Local traveltime operators

Generally speaking, there are two categories of methods exploiting traveltime information. One category refers to first-arrival time by traveltime, and traveltime tomography methods (Zelt & Chen 2016; Chen & Zelt 2017; Chen *et al.* 2017; Liao *et al.* 2017; Li *et al.* 2020) all belong to this category. The other category refers to a kinematic wavefront by traveltime, and traveltime-operator-based technologies (Hindriks & Duijndam 1999; Zhang *et al.* 2001; Jäger *et al.* 2001; Berkovitch *et al.* 2011; Sun *et al.* 2014; Sun *et al.* 2022), including the method discussed in this paper, all belong to this category. Although local traveltime operators have been discussed before (Hoecht *et al.* 2009; Buzlukov & Landa 2013; Bakulin *et al.* 2018; Bakulin *et al.* 2020; Sun *et al.* 2022), for the sake of discussion completeness, we briefly introduce this topic.

A local traveltime operator describes a seismic kinematic wavefront. Mathematically, it is a second-order operator with five unknown parameters (Sun *et al.* 2022):

$$\begin{aligned}\Delta t\,(x, y; x_0, y_0) &= t\,(x, y) - t\,(x_0, y_0)\\ &= A\cdot(x - x_0) + B\cdot(y - y_0)\\ &\quad + C\cdot(x - x_0)\cdot(y - y_0)\\ &\quad + D\cdot(x - x_0)^2 + E\cdot(y - y_0)^2, \quad (1)\end{aligned}$$

where $t(x, y)$ is the travel time of the trace located at (x, y) in a seismic gather, $t(x_0, y_0)$ is the travel time of the NLBF parameter trace located at (x_0, y_0) and the coefficients $\{A, B, C, D, E\}$ are the unknown NLBF parameters for a seismic kinematic wavefront centered at $t(x_0, y_0)$. In a rigorous mathematic language, these coefficients $\{A, B, C, D, E\}$ should be written as $\{A(x_0, y_0, t), B(x_0, y_0, t), C(x_0, y_0, t), D(x_0, y_0, t), E(x_0, y_0, t)\}$ since they are functions of both spatial location and time. However, for the convenience of writing, in the rest of this paper, we will just present them as $\{A, B, C, D, E\}$.

Note that the coordinates (x, y) and (x_0, y_0) shown in equation (1) should be treated as general coordinates as they are not limited to physical coordinates. For instance, if local traveltime operators are to be estimated for a 3D shot gather, then actual receiver coordinates can be used in equation (1); however, if local traveltime operators are to be estimated for a bunch of 2D shot gathers with the same receiver line, then we can use the source station number as the x coordinate and the receiver station number as the y coordinate in equation (1). The latter scheme also applies to cross-spread gathers.

For an input seismic gather, to estimate parameters $\{A, B, C, D, E\}$ of a local traveltime operator at (x_0, y_0, t), we need to maximize a semblance-based cost function defined

by

$$S\left(x_0, y_0\right) = \frac{\sum_{j=1}^{N}\left\{\sum_{i=1}^{M} u\left[x_i, y_i; t_j\left(x_0, y_0\right) + \Delta t\left(x_i, y_i; x_0, y_0\right)\right]\right\}^2}{M\sum_{j=1}^{N}\sum_{i=1}^{M}\left\{u\left[x_i, y_i; t_j\left(x_0, y_0\right) + \Delta t\left(x_i, y_i; x_0, y_0\right)\right]\right\}^2}, \quad (2)$$

where $u(x_i, y_i; t)$ represents a time sample of the trace located at (x_i, y_i) in the seismic data gather, M is the total amount of traces inside the spatial aperture of the local traveltime operator and N is the total amount of time samples within the temporal aperture of the local traveltime operator.

Equation (2) is a highly nonlinear function of parameters {A, B, C, D, E}, and several methods have been used as its solvers so far, including the 2+2+1 method (Bakulin *et al.* 2018), the 5D brute-force search method (Sun *et al.* 2022), the sequential dips-and-curvatures strategy (Bakulin *et al.* 2021), the adaptive simulated annealing (Ingber 1996), the efficiency-improved Genetic Algorithm (Acuna & Sun 2020) and the inpainting with deep neural networks (Gadylshin et al. 2020), among others. This paper focuses on the 2+2+1 method.

3. The 2+2+1 method and its CPU and GPU algorithms

The 2+2+1 method is a solver based on a local-search scheme that is commonly implemented as a part of the NLBF technology (Bakulin *et al.* 2018; Bakulin *et al.* 2021; Sun *et al.* 2022). We use figure 1 to explain this method in detail. We first parameterize the whole solution space by regularly-placed NLBF parameter traces marked by red dots in figure 1. Green dots represent seismic traces, and the red-circled black dot represents the NLBF parameter trace of interest, which the 2+2+1 method is working on. We refer to the coordinates of this NLBF parameter trace of interest by (x_0, y_0). Yellow areas in figure 1 represent spatial areas used for estimating different parameters of local traveltime operators in the 2+2+1 method. The 2+2+1 method first sets {B, C, E} at (x_0, y_0) to zeros and searches for the optimal values of {A, D} at (x_0, y_0) in a brute-force manner using all traces existing in its estimation aperture shown in figure 1a. Next, the parameters {B, E} at (x_0, y_0) are found in the same manner as {A, D} using traces in its estimation aperture shown in figure 1b. Finally, the 2+2+1 method fixes already estimated parameters {A, B, D, E} and finds the optimal value of {C} at (x_0, y_0) using traces falling into its estimation aperture via another brute-force search, as shown in figure 1c. Even though the 2+2+1 method is a local-search method, as it involves three sequential brute-force searches, should the solution space for parameters {A, B, C, D, E} be very big, its calculation cost is still very significant.

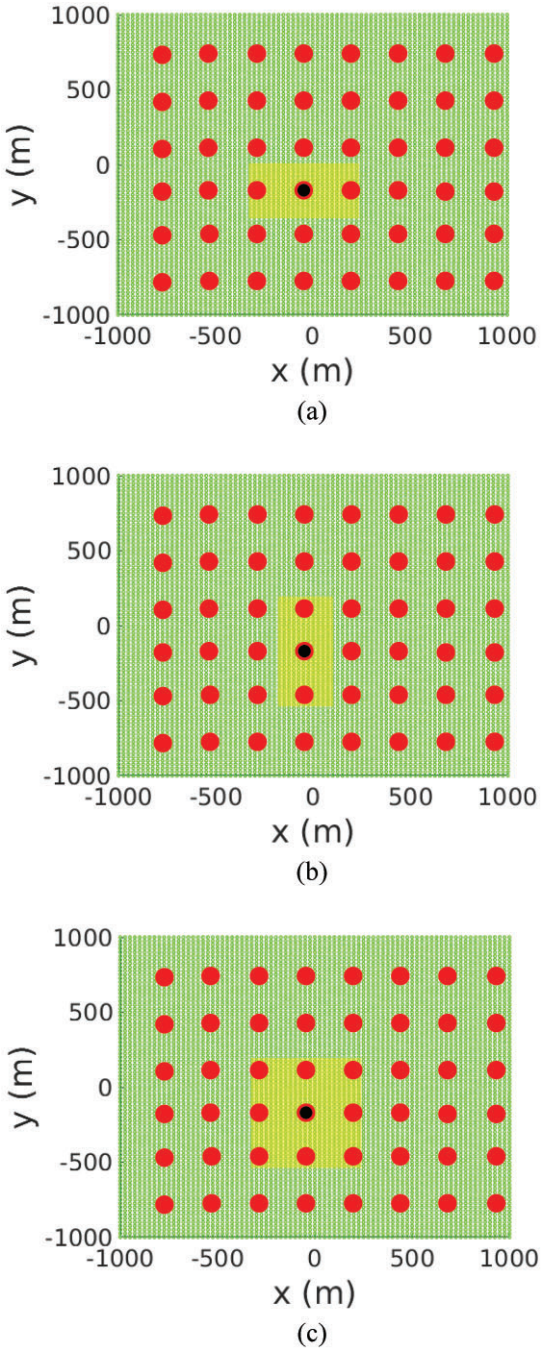

Figure 1. Spatial apertures for estimating various parameters: (a) {A, D}, (b) {B, E} and (c) {C}. Red dots mark NLBF parameter traces, and seismic traces are marked by green dots. The yellow area illustrates the spatial aperture for the local traveltime operator, and the red-circled black dot marks the target NLBF parameter trace.

Our CPU algorithmic implementation of the 2+2+1 method, shown in Table 1, has been successfully deployed in industrial seismic processing. It is based on a straightforward implementation of the method. For every NLBF parameter trace, we first collect traces falling into different apertures related to the estimation of parameters {C}, {A, D} and {B, E}. Next, for every time window of this NLBF parameter trace,

Table 1. The 2+2+1 CPU algorithm

Pseudocode for the 2+2+1 CPU Algorithm

```
Input: a seismic dataset, user-specified parameters to define the solution space for {A, B, C, D, E}
Output: {A, B, C, D, E, S} for each gather in the input dataset, where {S} is the semblance value
        corresponding to the parameters {A, B, C, D, E}
Use user-specified parameters to set search schemes for {A, D}, {B, E} and {C}
Set different apertures (shown in Figure 1) for {A, D}, {B, E} and {C}
foreach gather_i ∈ the input seismic dataset do
    Set locations of NLBF parameter traces for this gather;
    foreach y_i ∈ y direction of NLBF parameter traces do
        foreach x_i ∈ x direction of NLBF parameter traces do
            Collect traces falling into the aperture of {C};
            Collect traces falling into the aperture of {A, D};
            Collect traces falling into the aperture of {B, E};
            foreach t_i ∈ time direction of NLBF parameter traces do
                Set {B, C, E} to 0s and scan {A, D} in their solution space using traces in
                its aperture (Figure 1(a)) to pick out the combination that
                maximizes equation (2);
                Set {A, C, D} to 0s and scan {B, E} in their solution space using traces in
                its aperture (Figure 1(b)) to pick out the combination that
                maximizes equation (2);
                Fix {A, B, D, E} to the estimated values, and scan {C} in its solution space
                using traces in its aperture (Figure 1(c)) to pick out the value that
                maximizes the semblance value S defined by equation (2);
                Save these estimated {A, B, C, D, E, S} at (x_i, y_i, t_i);
            end
        end
    end
end
Output {A, B, C, D, E, S} of all NLBF parameter traces.
```

we estimate its corresponding local traveltime operators in three steps:

- Set {B, C, E} to zeros; scan all combinations of {A, D} in their solution space using traces falling into the aperture of {A, D}, as shown in figure 1a; the combination that maximizes our semblance-based cost function, i.e. equation (2), is picked out as the optimal pair {A, D} for the local traveltime operator at this time window of this NLBF parameter trace.
- Likewise, set {A, C, D} to zeros; scan all combinations of {B, E} in their solution space using traces falling into the aperture of {B, E}, as shown in figure 1b; the combination that maximizes our semblance-based cost function, i.e. equation (2), is picked out as the optimal pair {B, E} for the local traveltime operator at this time window of this NLBF parameter trace.
- Fix {A, B, D, E} at the previously estimated values; scan {C} in its solution space using traces falling into the aperture of {C}, as shown in figure 1c; the value that maximizes our semblance-based cost function, i.e. equation (2), is picked out as the optimal value {C} for the local traveltime operator at this time window of this NLBF parameter trace; the best semblance value is also saved as the final semblance value {S} corresponding to these estimated parameters {A, B, C, D, E}.

In Table 1, the part highlighted in green is parallelized via OpenMP directives for efficiency gains, i.e. the loop over the time direction in a NLBF parameter trace is carried out in parallel by different CPU threads.

From the perspective of hardware design, a GPU is for a completely different purpose from a CPU (NVIDIA 2020). A CPU is designed to be a low-latency device to excel at executing a sequence of operations as fast as possible. A GPU is designed to achieve an as-high-as-possible instruction throughput. To achieve this goal, a GPU pays the price of slower single-thread performance than a CPU. Furthermore, a GPU comes with a much larger memory bandwidth than a CPU to effectively support its higher instruction throughput. We design our NLBF 2+2+1 GPU algorithm to allow us to take full advantage of GPUs' high instruction throughput, and it is thus more complex than our CPU version. We adopt a hierarchic algorithm design, explained below:

- As data have to be organized and moved from the memory to GPU via CPU, and different parameter combinations, i.e. {A, D}, {B, E} and {C}, require different data, we first overlap data organization with data transfer

Table 2. The 2+2+1 GPU algorithm. The green parts are explained in Tables 3 and 4

Pseudocode for the 2+2+1 GPU Algorithm	
	Input: a seismic dataset, user-specified parameters to define the solution space for {A, B, C, D, E}, heap size and stream amount for the GPU
	Output: {A, B, C, D, E, S} for each gather in the input dataset, where {S} is the semblance value corresponding to the parameters {A, B, C, D, E}
1	Use user-specified parameters to set search schemes for {A, D}, {B, E} and {C};
2	Set different apertures (shown in Figure 1) for {A, D}, {B, E} and {C};
3	Set the user-specified heap size on the GPU;
4	Adaptively build the block and grid for the CUDA calculation;
5	Create a pool of CUDA streams;
6	**foreach** gather$_i$ ∈ the input seismic dataset **do**
7	Set locations of NLBF parameter traces for this gather;
8	**foreach** y_i ∈ y direction of NLBF parameter traces **do**
9	**foreach** x_i ∈ x direction of NLBF parameter traces **do**
10	Fetch an available CUDA stream from the stream pool;
11	Collect traces falling into the aperture of {A, D};
12	Asynchronously copy these traces in the aperture of {A, D} to the GPU, with the fetched stream as the stream identifier;
13	Collect traces falling into the aperture of {B, E};
14	Asynchronously copy these traces in the aperture of {B, E} to the GPU, with the fetched stream as the stream identifier;
15	Collect traces falling into the aperture of {C};
16	Asynchronously copy these traces in the aperture of {C} to the GPU, with the fetched stream as the stream identifier;
17	Run the 2+2+1 CUDA kernel to calculate {A, B, C, D, E, S} at (x_i, y_i),with the fetched stream as the stream identifier;
18	Asynchronously copy {A, B, C, D, E, S} at (x_i, y_i) from the GPU to the memory;
19	**end**
20	**end**
21	**end**
22	Output {A, B, C, D, E, S} of all NLBF parameter traces.

Table 3. The pseudocode for adaptively building the thread block and block grid for the CUDA kernel in the 2+2+1 GPU algorithm

Pseudocode for adaptively building the block and grid for the CUDA calculation	
	Input: solution dimensions N_A, N_B, N_C, N_D and N_E of parameters A, B, C, D and E; time dimension N_t of NLBF parameter traces
	Output: blockDim and gridDim, which are dim3 structures for the thread block and block grid
1	Set gridDim to (N_t, 1, 1);
2	$thread_{max} = max(N_A \cdot N_D, N_B \cdot N_E, N_C)$;
3	$thread_{max} = (thread_{max}\%32 == 0)?(thread_{max}):\left(\left\lfloor\frac{thread_{max}}{32}\right\rfloor * 32 + 32\right)$;
4	$thread_{max} = max(1024, thread_{max})$;
5	Set blockDim to ($thread_{max}$, 1, 1);
6	Output blockDim and gridDim.

using asynchronous copy. As different NLBF parameter traces create a natural opportunity for parallel computation for a GPU, we further exploit the concept of CUDA stream to overlap data transfer with GPU computation among different NLBF parameter traces. The 'foreach' loops in Table 2 show this top-level algorithm design in detail.

- As a GPU is designed as a natural vector-calculation machine, we need to unwrap 'for' loops in the CPU version as much as possible for efficiency gains. Considering that real input data may be of any size, we have to adaptively build thread blocks and block grids for our CUDA kernel. Table 3 shows the pseudocode for our adaptive design. We first parallelize the computational burdens of different time windows in an NLBF parameter trace via thread blocks. In other words, one thread block is used to deal with the local traveltime operator at a single time window of an NLBF

Table 4. The pseudocode for the CUDA kernel in the 2+2+1 GPU algorithm

Pseudocode for the CUDA kernel in the 2+2+1 GPU algorithm

Input: solution dimensions N_A, N_B, N_C, N_D and N_E of parameters A, B, C, D and E; traces in the apertures of {A, D}, {B, E} and {C}, respectively
Output: nonc
__shared__ float a_{best}, b_{best}, c_{best}, d_{best}, e_{best}, s_{best};
if threadIdx.x == 0 **do**
  Allocate memories for several float arrays: semblance[blockDim.x], a[blockDim.x], b[blockDim.x], c[blockDim.x], d[blockDim.x], e[blockDim.x];
end
__syncthreads();
semblance[threadIdx.x] = 0.0;
idx = threadIdx.x;
while idx < $N_A \cdot N_D$ **do**
  Get a_{thread} and d_{thread} corresponding to this idx;
  Evaluate equation (2) using { a_{thread}, 0, 0, d_{thread}, 0} for semb with the traces in the aperture of {A, D};
  if semblance[threadIdx.x] < semb **do**
    semblance[threadIdx.x] = semb;
    a[threadIdx.x] = a_{thread};
    d[threadIdx.x] = d_{thread};
  end
  idx = idx + blockDim.x;
end
__syncthreads();
if threadIdx.x == 0 **do**
  Pick out a_{best} and d_{best} corresponding to the best value in semblance[blockDim.x];
end
__syncthreads();
semblance[threadIdx.x] = 0.0;
idx = threadIdx.x;
while idx < $N_B \cdot N_E$ **do**
  Get b_{thread} and e_{thread} corresponding to this idx;
  Evaluate equation (2) using { 0, b_{thread}, 0, 0, e_{thread} } for semb with the traces in the aperture of {B, E};
  if semblance[threadIdx.x] < semb **do**
    semblance[threadIdx.x] = semb;
    b[threadIdx.x] = b_{thread};
    e[threadIdx.x] = e_{thread};
  end
  idx = idx + blockDim.x;
end
__syncthreads();
if threadIdx.x == 0 **do**
  Pick out b_{best} and e_{best} corresponding to the best value in semblance[blockDim.x];
end
__syncthreads();
semblance[threadIdx.x] = 0.0;
idx = threadIdx.x;
while idx < N_C **do**
  Get c_{thread} corresponding to this idx;
  Evaluate equation (2) using {a_{best}, b_{best}, c_{thread}, d_{best}, e_{best}} for semb with the traces in the aperture of {C};
  if semblance[threadIdx.x] < semb **do**
    semblance[threadIdx.x] = semb;
    c[threadIdx.x] = c_{thread};
  end
  idx = idx + blockDim.x;
end
__syncthreads();
if threadIdx.x == 0 **do**
  Pick out c_{best} corresponding to s_{best}, which is the best value in semblance[blockDim.x];
  Pass { a_{best}, b_{best}, c_{best}, d_{best}, e_{best}, s_{best} } to the global memory;
end
__syncthreads();
Return.

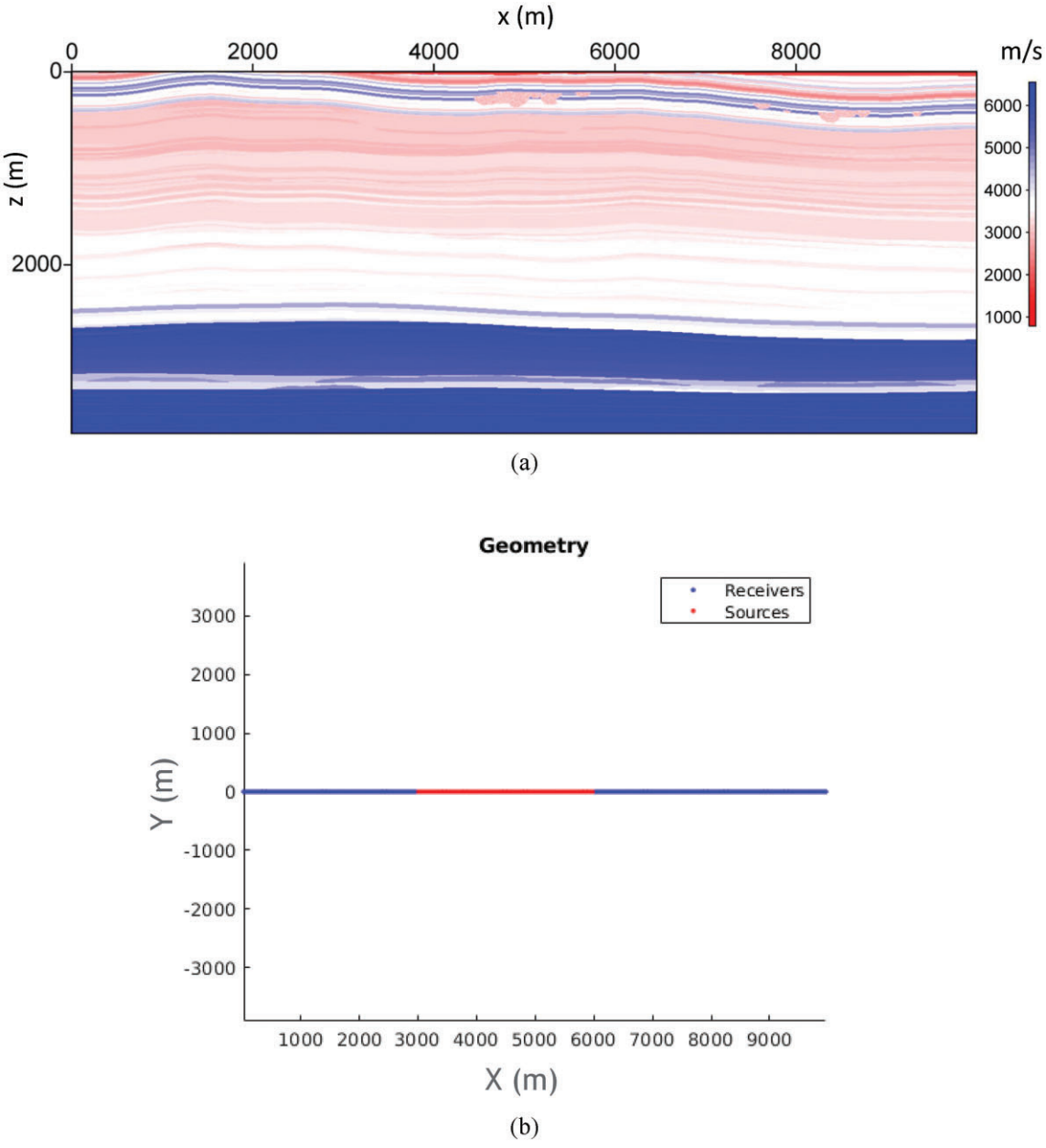

Figure 2. (a) The 2D synthetic model that is a slice from the 3D SEAM Arid velocity model. (b) Source and receiver distributions for the 2D synthetic dataset.

parameter trace. Considering we deal with seismic data, the limit of the x dimension of a block grid, which is 2^{31}-1 (NVIDIA 2020), is definitely more than enough for the time dimension of NLBF parameter traces, so the block grid can just be set as $(N_t, 1, 1)$, where N_t is the NLBF parameter trace's time dimension. One thread evaluates one parameter combination for parallel computation inside each thread block. However, since the maximum number of threads in a block can only be 1024 due to the limitation of hardware design (NVIDIA 2020), while our solution dimensions are unpredictable as they depend on users' choices, we then have to build our thread-block size, $thread_{max}$, adaptively: first take the largest solution dimension among $N_A \cdot N_D$, $N_B \cdot N_E$ and N_C (where N_A, N_B, N_C, N_D and N_E are the solution dimensions of parameters A, B, C, D and E, respectively) as its trial value, i.e. $thread_{max} = max(N_A \cdot N_D, N_B \cdot N_E, N_C)$; next, round $thread_{max}$ to the least integer multiple of 32 larger than $thread_{max}$ (ceiling), where 32 is the hardware-defined warp size of GPUs (NVIDIA 2020); finally, subject to the limitation of the maximum 1024 threads in a block, take the larger value between the current $thread_{max}$ and 1024 as the final thread-block size, i.e. $thread_{max} = max(thread_{max}, 1024)$.

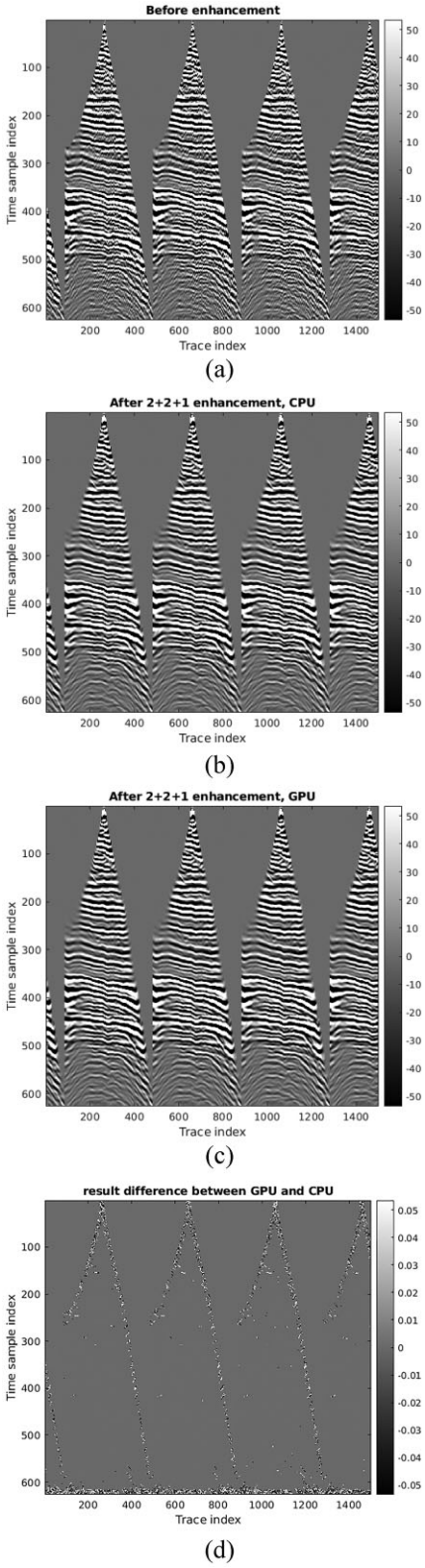

Figure 3. (a) NMO-corrected raw gathers from several receiver lines (b) and (c) show the same gathers after enhancement by NLBF with the 2+2+1 method implemented on CPU and GPU, respectively. (d) Data difference between the CPU and GPU result plotted using a different scale from (a)–(c).

- The CUDA kernel for our 2+2+1 method directly estimates parameters {A, B, C, D, E, S} for one local traveltime operator at a single time window of an NLBF parameter trace, and this is the low-level algorithm design of our NLBF 2+2+1 GPU algorithm. With all the data pertaining to different parameters available in the GPU memory, each thread in the block evaluates the cost function, i.e. equation (2), using either a particular parameter combination (for {A, D} and {B, E}) or a particular parameter value (for {C}), stores the corresponding semblance value in a specific array location corresponding to this thread, and finally asks the 0th thread in the block to pick out the best semblance value and its corresponding parameter combination (for {A, D} and {B, E}) or parameter value (for {C}). Please note that the best semblance value from the parameter {C} estimation is the final semblance value {S} corresponding to this set of estimated parameter values {A, B, C, D, E}. Eventually, this 0th thread passes the estimated parameters {A, B, C, D, E, S} to the GPU memory. The pseudocode of our CUDA kernel is shown in Table 4, with all its technical details presented.

To summarize, the CPU algorithmic implementation is a one-to-one translation of the 2+2+1 method, where the calculation is parallelized in the time direction of NLBF parameter traces via OpenMP directives to achieve efficiency gains. In contrast, the GPU algorithmic implementation of the 2+2+1 method takes the best advantage of a GPU's high instruction throughput by exploiting different inherent GPU features: it uses asynchronous copy to overlap CPU data organization with data transfer from CPU to GPU; it uses CUDA streams to parallelize computations on different NLBF parameter traces; it uses thread blocks to calculate local traveltime operators of NLBF parameter traces and it uses threads to evaluate equation (2) over different concrete solution spaces.

4. Examples

In this section, we use one synthetic and one challenging single-sensor field dataset, which were already described in other NLBF-related publications (Sun *et al.* 2022), to demonstrate the efficiency gains achieved by our 2+2+1 GPU algorithm over the CPU version on estimating local traveltime operators in NLBF. The hardware environment running the 2+2+1 CPU version comprises two Xeon Gold 6136 CPUs (3.00 GHz, 12 cores), and all 24 cores are actively engaged via OpenMP directives in our tests. The 2+2+1 GPU version runs on a single NVIDIA Tesla V100 GPU with 32 GB onboard memory, and only one core of a Xeon Gold 6142 CPUs (2.60 GHz, 16 cores) is involved. The operating

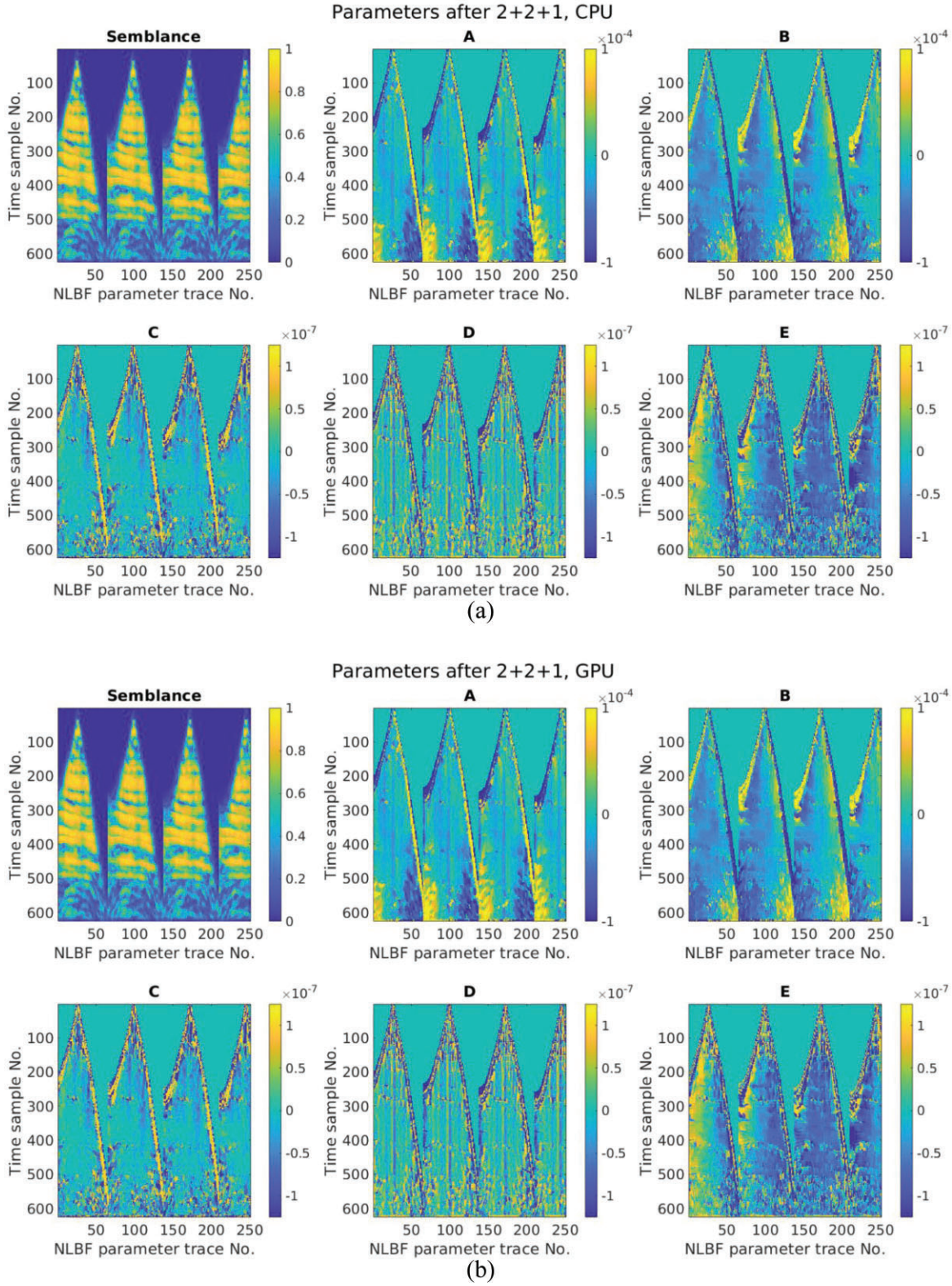

Figure 4. (a) and (b) Semblance values and parameters {A, B, C, D, E} for some selected NLBF parameter traces estimated by the 2+2+1 CPU and GPU versions, respectively. (c) Difference panels between the CPU and the GPU result plotted using different scales from (a) and (b)

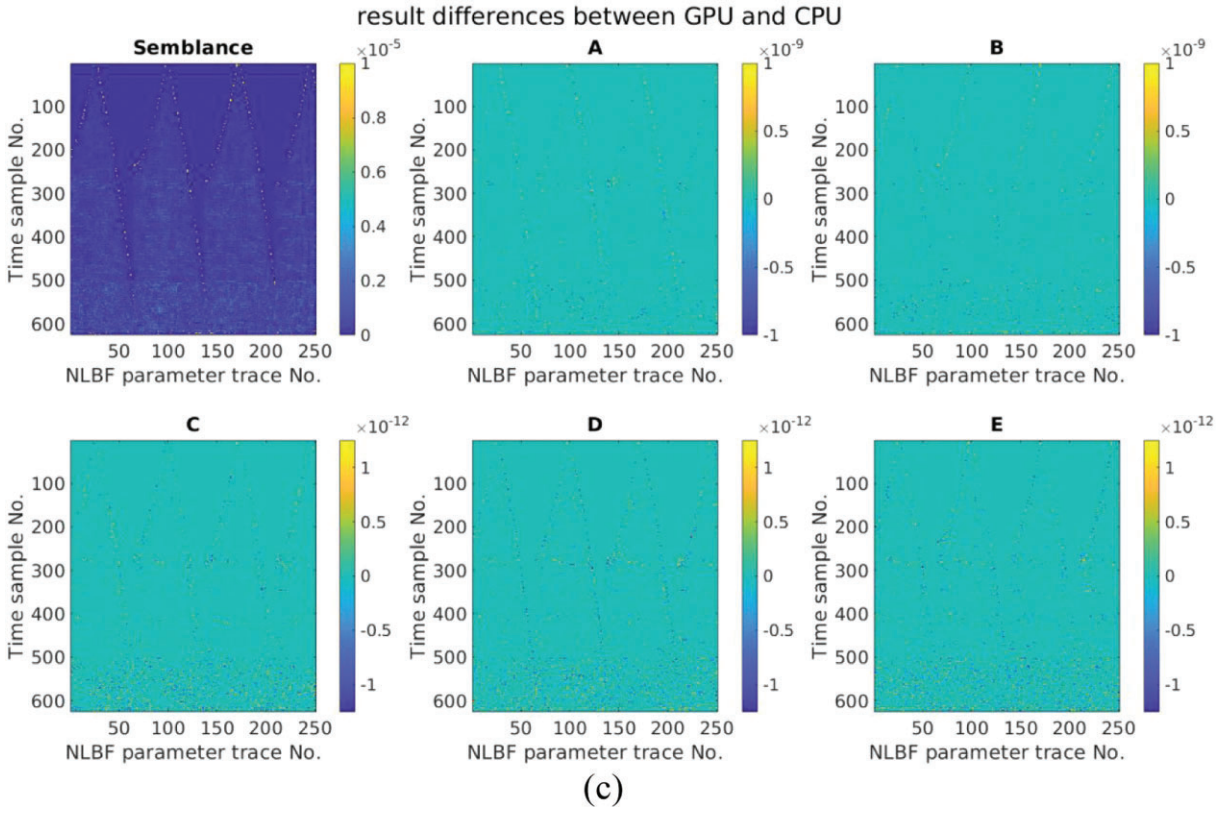

Figure 4. Continued.

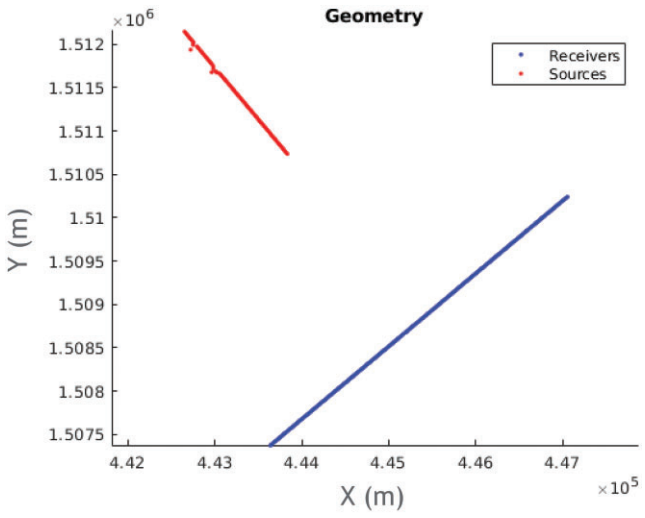

Figure 5. Source and receiver distributions for the challenging 3D single-sensor field dataset.

system running on our CPU (GPU) platform is Red Hat 7.x (CentOS 7.x). Both the CPU version and the GPU version of the 2+2+1 method are coded in C++. However, due to some practical limitations, we use the following compilers: Intel Parallel Studio XE 2017 Update 2 for the CPU version and CUDA 9.0 Update 4 jointly with Intel Parallel Studio XE 2017 Update 4 for the GPU version.

Table 5. Run-time comparison between the 2+2+1 CPU version and the 2+2+1 GPU version on the synthetic dataset

Run no./scheme	2+2+1 GPU (unit: s)	2+2+1 CPU (unit: s)
1	5.87	45.70
2	5.88	45.67
3	5.98	45.64
4	5.66	45.73
5	5.84	45.76
6	5.68	45.55
7	6.04	45.67
8	5.94	45.69
9	5.78	45.45
10	5.90	45.67
Average run-time	5.86	45.65

Regarding specific settings for parameters {A, B, C, D, E} of local traveltime operators and spacing between NLBF parameter traces, these have been discussed in Bakulin *et al.* (2020). For discussion completeness, hereby we briefly introduce our rules of thumb to define them in practice. Actual parameter values are controlled by geology and are empirically set. Generally speaking, parameters {A, B} are in the range of $\pm 10^{-5}$ s m^{-1}, and parameters {C, D, E} are in the range of $\pm 10^{-7}$ s m^{-2}. Regarding spacing between NLBF parameter traces, it is dependent on the estimation aperture size, and our empirical choice for the spacing is between 0.30

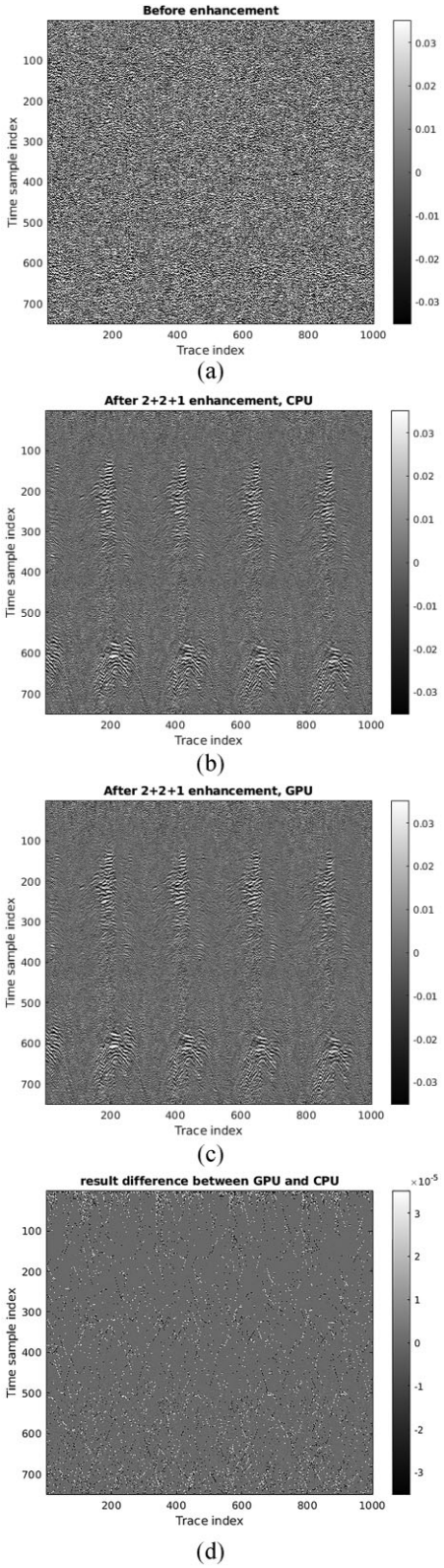

Figure 6. (a) Common-shot gathers from several receiver lines extracted from the raw data. (b) and (c) The same data but after enhancement with NLBF using the 2+2+1 CPU version and the GPU version, respectively. (d) Data difference between the CPU and the GPU result plotted using a different scale from (a)–(c).

and 0.70 times of the estimation aperture size of parameter {C}, as shown in figure 1.

Our synthetic example is generated using a 2D synthetic model, a slice from the 3D SEAM Arid velocity model (Oristaglio 2015; Sun *et al.* 2022). Figure 2 displays the model and source and receiver distributions for this synthetic dataset. The data simulation is done via a finite-difference engine. The grid spacing is 0.625 × 0.625 m, with both sources and receivers located at $z = 0.625$ m. No surface-related multiples exist in the dataset due to the adoption of an absorption surface. Both source- and receiver-interval are 25 m, and a Ricker wavelet with a dominant frequency of 20 Hz is used as a source. The time sampling rate is 4 ms, and the record length is 2.5 s. Before running our 2+2+1 method for estimating local traveltime operators, we first apply NMO correction to this dataset making reflection events flattish. Figure 3a shows several raw gathers after the NMO correction. We can easily see some scattering noise in the data. For this dataset, we use the receiver location as the x coordinate and the source location as the y coordinate for local traveltime operators. The spacing between NLBF parameter traces is 140 m in both x and y directions. The spatial apertures in x and y dimensions for different parameters are as follows: 400 × 35 m for {A, D}, 35 × 400 m for {B, E}, and 400 × 400 m for {C}. Using the format [min : step : max] to denote the search space and the search step for each parameter, the actual search schemes are $[-10^{-4} : 10^{-5} : 10^{-4}\ \mathrm{s\ m^{-1}}]$ for {A, B} and $[-1.25{\times}10^{-7} : 0.25{\times}10^{-7} : 1.25{\times}10^{-7}\ \mathrm{s\ m^{-2}}]$ for {C, D, E}. Figure 4 shows the estimated semblance values and parameters {A, B, C, D, E} for some selected NLBF parameter traces from both the CPU version and the GPU version of our 2+2+1 method. Figure 4 parts a and b are visually identical but only exhibit minor differences at the floating-number level, as shown in figure 4c. Such minor deviations are reasonable as these floating results are calculated using different hardware platforms with distinct software environments. To appreciate the efficiency gain achieved by the GPU version over the CPU version and avoid the influence of system jitter, we run each version 10 times on this example and take their average run time for a comparison. Table 5 lists the detailed information for these 10 runs. In this example, our 2+2+1 GPU version gains a speed-up factor of 7.8 over our 2+2+1 CPU version. We further apply NLBF with these estimated local traveltime operators to enhance the data (Sun *et al.* 2022) shown in figure 3a, and the corresponding enhanced results from the CPU version and the GPU version are shown in figure 3 parts b and c, respectively. We can clearly see the image quality improvement as those scattering noise have been nicely suppressed. At the same time, all major reflection events have been preserved with very high fidelity. Similar to figure 4, figure 3 parts b

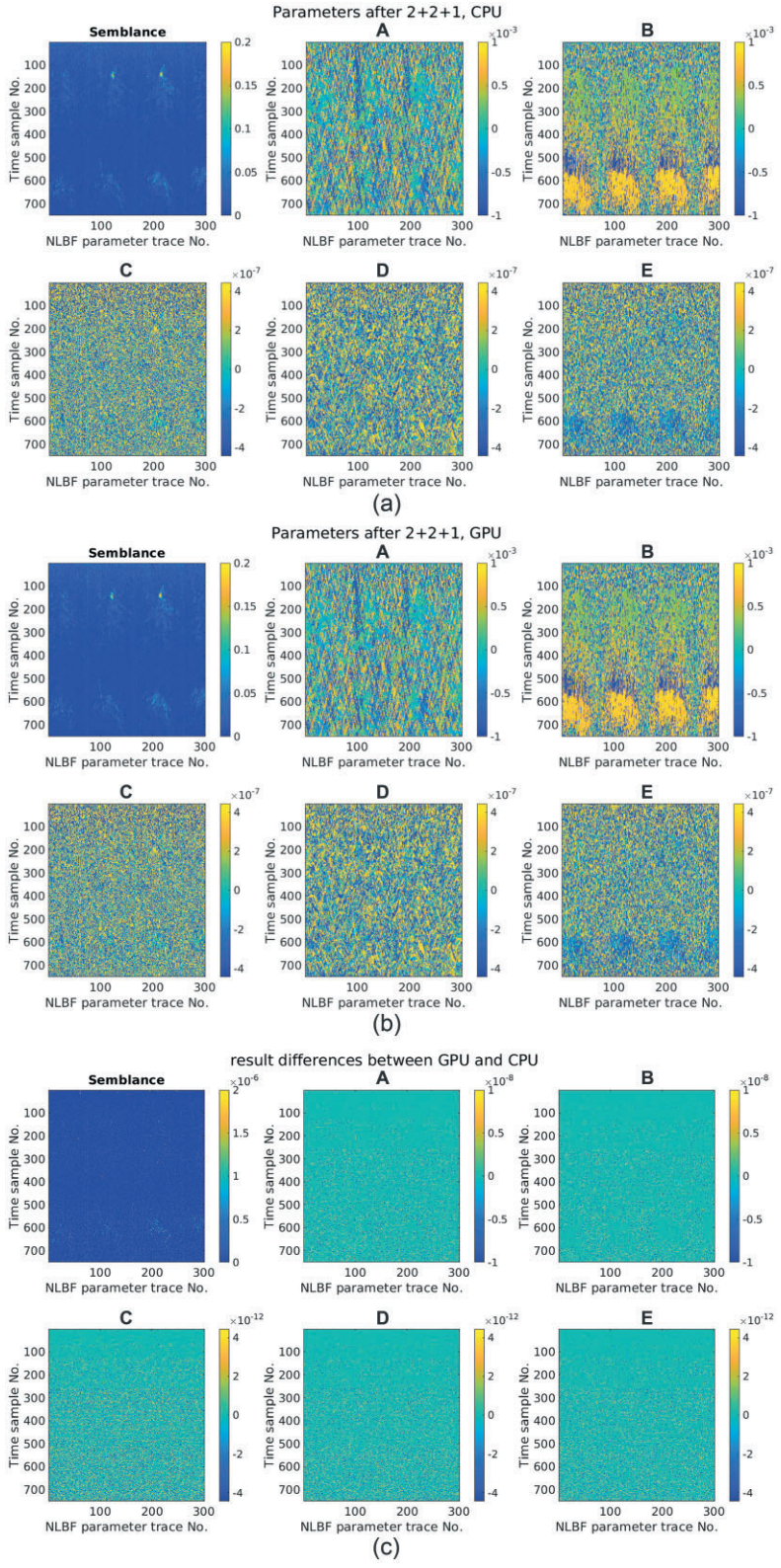

Figure 7. (a) and (b) Semblance values and parameters {A, B, C, D, E} for some selected NLBF parameter traces in the field data example estimated by the 2+2+1 CPU version and the GPU version, respectively. (c) Difference panels between the CPU and the GPU result plotted using different scales from (a) and (b).

Table 6. Run-time comparison between the 2+2+1 CPU version and the 2+2+1 GPU version on the challenging 3D field dataset

Run no./scheme	2+2+1 GPU (unit: s)	2+2+1 CPU (unit: s)
1	146.47	1643.79
2	148.36	1641.00
3	146.59	1638.91
4	148.45	1638.96
5	145.87	1639.02
6	147.27	1639.00
7	147.95	1640.70
8	147.86	1639.45
9	147.89	1639.70
10	147.19	1638.82
Average run-time	147.39	1639.94

and c are visually identical, despite minor differences at the floating-number level, as shown in figure 3d.

Our field data example is an extremely challenging 3D single-sensor dataset acquired in a desert environment. Figure 5 shows its source and receiver distributions. This dataset is sorted for enhancement in the cross-spread domain, which is defined as all traces with their sources from the same source line and their receivers from the same receiver line (Li 2008). In total, there are 94 sources and 224 receivers, and the spacing for both sources and receivers is 20 m. This field dataset is heavily contaminated by near-surface scattering noise, which can be appreciated in figure 6a. We do not observe any coherent events on these raw data gathers. For this dataset, we use the receiver station number as the x coordinate and the shot station number as the y coordinate in equation (1). The spacing between NLBF parameter traces is 50 m in both the x and y directions. The spatial apertures in x and y dimensions for different parameters are as follows: 300×35 m for {A, D}, 35×300 m for {B, E}, and 300×300 m for {C}. The actual search schemes for different parameters are $[-10^{-3}: 1.33{\times}10^{-5} : 10^{-3}$ s m$^{-1}]$ for {A, B} and $[-4.44{\times}10^{-7} : 4.44{\times}10^{-8} : 4.44{\times}10^{-7}$ s m$^{-2}]$ for {C, D, E}. Figure 7 parts a and b show the estimated semblance values and parameters {A, B, C, D, E} of some selected NLBF parameter traces from both the 2+2+1 CPU version and the 2+2+1 GPU version. Table 6 lists the detailed information of 10 different runs for a run-time comparison. We can see that the 2+2+1 GPU version gains a speed-up factor of 11.1 over the 2+2+1 CPU version. We also apply NLBF with the estimated local traveltime operators to enhance data from figure 6a. Figure 6 parts b and c show the corresponding enhanced results. Similar to figures 3 and 4, results from both the CPU and the GPU version in this field data example are also visually identical, even though they differ slightly at the floating-number level, as shown in figures 6d and 7c.

5. Conclusions

Our paper focuses on the 2+2+1 method for estimating local traveltime operators in the nonlinear beamforming technology and comprehensively introduces its algorithmic implementations on both the CPU platform and the GPU platform. We exploit features of graphics cards and the CUDA programming language, including asynchronous data copy, CUDA streams, block grids, thread blocks etc., to fully parallelize numerical evaluations of the semblance-based cost function in the nonlinear beamforming technology. We use a 2D synthetic example and a challenging 3D single-sensor field dataset acquired in a desert environment to demonstrate a speed-up factor of ~10 achieved by our 2+2+1 GPU algorithm over the 2+2+1 CPU algorithm. The considerable efficiency improvement from the 2+2+1 GPU algorithm greatly benefits the nonlinear beamforming technology, which can help seismic data processing and interpretation in challenging areas.

Acknowledgements

The Top 500 was retrieved August 27, 2021, from https://www.top500.org/. For computer time, this research used the resources of the Supercomputing Laboratory at King Abdullah University of Science & Technology (KAUST) in Thuwal, Saudi Arabia.

Data availability statement

Research data are not shared.

Conflict of interest statement. None declared.

References

Acuna, Y.P.V. & Sun, Y., 2020. An efficiency-improved genetic algorithm and its application on multimodal functions and a 2D common reflection surface stacking problem, *Geophysical Prospecting*, **68**, 1189–1210.

Bakulin, A., Silvestrov, I., Dmitriev, M., Neklyudov, D., Protasov, M., Gadylshin, K. & Dolgov, V., 2020. Nonlinear beamforming for enhancement of 3D prestack land seismic data, *Geophysics*, **85**, V283–V296.

Bakulin, A., Silvestrov, I., Dmitriev, M., Neklyudov, D., Protasov, M., Gadylshin, K., Tcheverda, V. & Dolgov, V., 2018. Nonlinear beamforming for enhancing prestack seismic data with a challenging near surface or overburden, *First Break*, **36**, 121–126.

Bakulin, A., Silvestrov, I. & Protasov, M., 2021. Evaluating strategies for estimation of local kinematic parameters in noisy land data: quality versus performance trade-offs, *Journal of Geophysics and Engineering*, **18**, 890–907.

Berkovitch, A., Deev, K. & Landa, E., 2011. How nonhyperbolic multifocusing improves depth imaging, *First Break*, **29**, 95–103.

Buzlukov, V. & Landa, E., 2013. Imaging improvement by prestack signal enhancement, *Geophysical Prospecting*, **61**, 1150–1158.

Chen, J. & Zelt, C. A., 2017. Comparison of full wavefield synthetics with frequency-dependent traveltimes calculated using wavelength-dependent velocity smoothing, *Journal of Environmental and Engineering Geophysics*, **22**, 133–141.

Chen, J., Zelt, C. A. & Jaiswal, P., 2017. Detecting a known near-surface target through application of frequency-dependent traveltime tomography and full-waveform inversion to P- and SH-wave seismic refraction data, *Geophysics*, **82**, R1–R17.

Elsen, E., LeGresley, P. & Darve, E., 2008. Large calculation of the flow over a hypersonic vehicle using a GPU, *Journal of Computational Physics*, **227**, 10148–10161.

Gadylshin, K., Silvestrov, I. & Bakulin, A., 2020. Inpainting of local wavefront attributes using artificial intelligence for enhancement of massive 3-D pre-stack seismic data, *Geophysical Journal International*, **223**, 1888–1898.

Hindriks, C. & Duijndam, A., 1999. Handling near-surface effects in imaging by using common focal point technology, *SEG Expanded Abstract*, 575–578.

Hoecht, G., Ricarte, P., Bergler, S. & Landa, E., 2009. Operator-oriented CRS interpolation, *Geophysical Prospecting*, **57**, 957–979.

Ingber, L., 1996. Adaptive simulated annealing (ASA): lessons learned, *Control and Cybernetics*, **25**, 32–54.

Jäger, R., Mann, J., Höcht, G. & Hubral, P., 2001. Common-reflection-surface stack: images and attributes, *Geophysics*, **66**, 97–109.

Klöckner, A., Warburton, T., Bridge, J. & Hesthaven, J.S., 2009. Nodal discontinuous Galerkin methods on graphics processors, *Journal of Computational Physics*, **228**, 7863–7882.

Li, Ch., Liu, J., Liao, J. & Husthouse, A., 2020. 2D high-resolution crosswell seismic traveltime tomography, *Journal of Environmental and Engineering Geophysics*, **25**, 47–53.

Li, F., Gao, J., Jiang, X. & Sun, W., 2018. A causal imaging condition for reverse time migration using the discrete Hilbert transform and its efficient implementation on GPU, *Journal of Geophysics and Engineering*, **16**, 894–912.

Li, X., 2008. An introduction to common offset vector trace gathering, *CSEG Recorder*, **33**, 28–34.

Liao, J., Guo, Z.W., Liu, H.X., Dai, S.X., Zhao, Y.L., Wang, L.X., Wang, H.Z. & Hursthouse, A., 2017. Application of frequency-dependent traveltime tomography to 2D crosswell seismic field data, *Journal of Environmental and Engineering Geophysics*, **22**, 421–426.

Liu, M. & Grana, D., 2019. Accelerating geostatistical seismic inversion using tensor flow: a heterogeneous distributed deep learning framework, *Computers and Geosciences*, **124**, 37–45.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H. & Teller, E., 1953. Equation of state calculations by fast computing machines, *The Journal of Chemical Physics*, **21**, 1087–1092.

NVIDIA, 2020. *CUDA C++ Programming Guide*. https://docs.nvidia.com/cuda/archive/11.1.0/cuda-c-programming-guide/index.html.

Oristaglio, M., 2015. SEAM Update, *The Leading Edge*, **34**, 466–468.

Storn, R. & Price, K., 1997. Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces, *Journal of Global Optimization*, **11**, 341–359.

Shi, Y. & Wang, Y., 2016. Reverse time migration of 3D vertical seismic profile data, *Geophysics*, **81**, S31–S38.

Sun, Y., Silvestrov, I. & Bakulin, A., 2022. Enhancing 3D land seismic data using nonlinear beamforming based on the efficiency-improved genetic algorithm, *IEEE Transactions on Evolutionary Computation*, doi: 10.1109/TEVC.2022.3149579.

Sun, Y., Tonellet, T., Kamel, B. & Bakulin, A., 2016. A 2D automatic converted-wave statics correction method, *The Leading Edge*, **35**, 280–284.

Sun, Y., Tonellet, T., Kamel, B. & Bakulin, A., 2017. A two-phase automatic static correction method, *Geophysical Prospecting*, **65**, 711–723.

Sun, Y. & Verschuur, D.J., 2014. A self-adjustable input genetic algorithm for the near-surface problem in geophysics, *IEEE Transactions on Evolutionary Computation*, **18**, 309–325.

Sun, Y., Verschuur, E. & Vrolijk, J.W., 2014. Solving the complex near-surface problem using 3D data-driven near-surface layer replacement, *Geophysical Prospecting*, **62**, 491–506.

Zelt, C. A. & Chen, J., 2016. Frequency-dependent traveltime tomography for near-surface refraction data, *Geophysical Journal International*, **207**, 72–88.

Zhang, Y., Bergler, S. & Hubral, P., 2001. Common-reflection-surface (CRS) stack for common offset, *Geophysical Prospecting*, **49**, 709–718.